A FORMAL DESCRIPTION OF A SUBSET OF ALGOL

John McCarthy

Stanford University

Prepared for:

Advanced Research Projects Agency

24 September 1964

STANFORD ARTIFICIAL INTELLIGENCE PROJECT September 24, 1964
Memo No. 24

A FORMAL DESCRIPTION OF A SUBSET OF ALGOL

by John McCarthy

Abstract: We describe Microalgol, a trivial subset of Algol, by means of an interpreter. The notions of abstract syntax and of "state of the computation" permit a compact description of both syntax and semantics. We advocate an extension of this technique as a general way of describing programming languages.

The research reported here was supported in part by the Advanced Research Project Agency of the Office of the Secretary of Defense (SD-183)

A FORMAL DESCRIPTION OF A SUBSET OF ALGOL

by John McCarthy

1. Introduction:

In my paper Towards a Mathematical Science of Computation, Proceedings of the ICIP, 1962, I advocated defining programming languages in the following way:

1 - Give the syntax in an abstract analytic form, i.e. for each type of expression name the predicates for telling how it is composed and for getting its parts.

2 - The abstract syntax makes no commitments about how sums, products, etc. are actually represented by symbolic expressions. To define a concrete syntax one represents the abstract syntactic predicates and functions by functions of strings.

3 - Next one defines what information is included in describing the state of the computation, e.g. this includes the values currently assigned to the program variables.

4 - Then one describes the semantics of the language by defining a function $\xi' = \text{lang}(\pi, \xi)$ that gives the state ξ' that results from applying the program π to the state ξ.

Our object in this paper is to carry this procedure out for a very small subset of Algol called Microalgol. This will illustrate the method in an easy case; all the difficult aspects of Algol are eliminated.

2. Informal Description of Microalgol:

Microalgol is a language for programming about, not for programming in. It has no declarations and no arrays, and the only statements are assignments and conditional go to 's of the form if p then go to a.

In forming the right sides of assignment statements one may use sums, products, differences, quotients and conditional expressions involving the relational operators = and <. All arithmetic operators take two operands. Here is an example of a Microalgol program

```
            root: = 1   ;
a:  root: = 0.5 x(root + x/root) ;
    error: = root x root -x;
    perror: = if error > 0.0 then error else 0.0 - error;
    if perror > .00001 then go to a;
```

3. Abstract Analystic Syntax of Microalgol:

We shall first give the abstract analytic syntax of the terms that can appear on the right sides of assignment statements. It is given by the following table:

predicate	associated functions		examples
isvar (τ)			x
isconst(τ)	val (τ)	N.B. This is a semantic function	.001
issum (τ)	addend (τ)	augend (τ)	a + x x y
isdiff (τ)	subtrahend (τ)	minuend (τ)	a - x x y
isprod (τ)	multiplier (τ)	multiplicand (τ)	x x (a+b)
alquotient (τ)	numerator (τ)	denominator (τ)	x/root
iscond (τ)	proposition (τ)	antecedent(τ)consequent(τ)	if x < 3 then y else 2
isequal (τ)	lefteq (τ)	righteq (τ)	x = 3
isless (τ)	lefl (τ)	rightl (τ)	x < 3

The idea, taken from the ICIP paper is that any term is of one of the eight types and that the predicates enable us to tell which. Once the type is decided the syntactic functions associated with that type are defined and give us the parts of the expression. Thus if τ is a + x x y then issum (τ) is true and addend (τ) is a and augend (τ) is x x y.

Now we give the abstract syntax of Microalgol statements:

predicate	associated functions		examples
assignment (s)	left (s)	right (s)	s is "root:=0.5x(root+x/root)" left(s) is "root" right (s) is "0.5x(root+x/root)"
goto (s)	proposition(s)	destination(s)	s is "if perror> .00001 then go to a" proposition (s) is "perror > .00001" destination (s) is "a"

Finally, we give the abstract syntax of Microalgol programs:

1. If π is a program and η is a statement number then statement (π, η) is the ηth statement of the program. Thus for the program we have been using as an example, statement $(\pi, 3)$ is "error: = root x root - x".

2. If ℓ is a label than numb (ℓ, π) is the statement number corresponding to the label if there is one. Thus, in our program, numb (a, π) is 2.

3. The predicate end (π, η) is true if there is no ηth statement. Thus end$(\pi,6)$ is true in our example.

4. The States of Microalgol

The state of a Microalgol computation is given by a state vector ξ which tells us the value currently assigned to each variable and also the statement number about to be executed. We shall treat the statement number as a pseudo-variable called sn.

Associated with state vectors are two functions:

1. c(var;ξ) gives the value assigned to the variable var in state ξ.

2. a(var, value, ξ) gives the new state that results from the state ξ when the number value is assigned to the variable var.

Some of the properties of state vectors are given in the ICIP paper.

The values of Microalgol variable are real numbers. Of course, only small integers can ever turn up as values of sn.

5. The Semantics of Microalgol

The semantics of Microalgol is given by a recursively defined function micro (π, ξ) that gives the state in which a Microalgol program π will terminate if it is entered in state ξ. First, however, we give the function value (τ , ξ) that tells us the value of a term. We have

value (τ , ξ) = if isvar (τ) then c(τ,ξ)
else if isconst (τ) then val (τ)
else if issum (τ) then value (addend (τ), ξ) + value (augend (τ), ξ)
else if isdiff (τ) then value (subtrahend (τ), ξ) - value (minuend (τ), ξ)
else if isprod (τ) then value (multiplier (τ), ξ) x value (multiplicand (τ), ξ)
else if isquotient (τ) then value (numerator (τ), ξ)/value (denominator(τ),ξ)
else if iscond (τ) then (if value (proposition (τ), ξ) then
value(antecedent (τ), ξ) else value (consequent (τ), ξ))
else if isequal (τ) then (value(lefteq (τ), ξ) = value (righteq (τ), ξ))
else if isless (τ) then (value (leftl (τ), ξ) < value (rightl(τ), ξ))

For definiteness we shall assume that the arithmetic in Microalgol as expressed by the operators + - x / = < is real number arithmetic. Little would be changed, however, if we restricted the values of variables to numbers represented in some machine and meant by the operators the operations of the machine.

We now can write

micro (π , ξ) = $(\lambda$ n .if end (π , n) then ξ
else $(\lambda$ s . if assignment (s) then
micro(π, a(sn, n + 1, a(left(s), value (right(s),ξ),ξ)))
else if goto (s) then
micro (π, a(sn, if value (proposition (s), ξ) then
numb (destination (s),π) else n + 1, ξ)))
(statement (n,π)) (c (sn,ξ))

This completes the description of abstract Microalgol. In order to describe a concrete Micro-algol it is only necessary to represent the abstract sytactic predicates and functions by predicates and functions on strings.

5. Two Concrete Realizations of Abstract Microalgol

We shall present two realizations of Microalgol. This first is a Lisp S-expression realization suitable for use inside a machine and the second corresponds to the concrete syntax of ALGOL 60.

A LISP realization called LMA.

We use LISP expressions for the terms as follows:

a. atoms for variables
b. Lisp numbers for constants

c. (PLUS α b) for sums
d. (DIFF α β)
e. (TIMES α β)
f. (RATIO α β)
g. (IF α β γ)
h. (EQUALS α β)
i. (LESS α β)
j. (ASSIGN α β)
k. (GO α β)

2. We represent a program by a list of its statements leaving a place for the label that is left blank if there is no label. The syntactic functions are as follows:

a. isvar (τ) = atom [τ] $\wedge$ $\neg$ numberp [τ]
b. isconst (τ) = numberp [τ]
c. issum (τ) = eq [car [τ]; PLUS]
addend (τ) = cadr [τ]
augend (τ) = caddr [τ]
d. isprod (τ) = eq [car [τ]; TIMES]
multiplier (τ) = cadr [τ]
multiplicand (τ) = caddr [τ]

We omit the obvious for isdiff, isquot, iscond, iseq, isless, assign, goto.

statement(n, π) = if n = 1 then cadar [π] else statement [n-1;cdr[π]]
numb (a, π) = if eq [car [π]; a] then 1 else 1 + numb [a,cdr [π]]
end (π, η) = null [π] $\vee$ [$\eta > 1 \wedge$ end [cdr [π]; η-1]]

6. A Standard Realization

In order to describe a realization of Microalgol that corresponds to ALGOL 60 we need to compute with strings of symbols. For this purpose we shall use the linear LISP of [3] and we only give a few of the syntactic predicates, namely, statement (η, π), issum (τ), addend (τ) and isprod (τ).

statement (η, π) = if η = 1 then delim (";", π) else statement (η-1, strip (";", π))

delim (a, π) = if first (π) = a then $\wedge$ else prefix (first (π), delim (a, rest (π)))

strip (a, π) = if first (π) = a then rest (π) else strip (a, rest (π))
issum (τ) = isop (τ, 0,"+")
isop (τ, η, a) = if null (τ) then F else if
first (π) = "(" then isop (
rest (τ), η+1,a) else if first (π) = ")" then isop (rest (τ), η-1,a)

```
else if η > 0 then isop (rest (τ), η,a) else if first (τ) = a then T
    else isop (rest (τ), η,a)
    augend (τ) = deparen( delim 1 ( "+", 0, τ))
    delim 1 (a, η, τ) = if first (τ) = "(" then (prefix (
first (τ), delim 1 (a, η + 1, rest (τ)))) else if first (τ) = ")"
then prefix (first (τ), delim 1 (a, η- 1, rest (τ))) else if
η > 0 then prefix ( first (τ), delim 1 (a, η , rest (τ))
else if first (τ) = a then Λ else
prefix (first (τ), delim 1 (a, η, rest (τ)))
deparen (τ) = if ¬ first (τ) =    "(" then τ else
dep 1 (rest (τ))
      dep 1 (τ) = if ¬ null (rest (τ)) then prefix (first (τ),
dep 1 (rest (τ))) else Λ
isprod (τ) = ¬ issum (τ) ∧ ¬ isdiff (τ) ∧ isop (τ , 0, "x")
```

7. What about ALGOL

The semantics of Microalgol was described entirely by the two formulas of section 5. Algol is considerably more complicated, but it will be relatively easy to write down the functions once we have decided what goes into the state. The following complications arise.

1. We have to be able to describe the situation in which a term is partially evaluated in order to describe the state during the execution of a type procedure.

2. The chain of procedure entries must be described including the recursive entries.

3. The current declarations and those associated with higher levels of recursion must be included.

4. Call-by-name parameters require the association of expressions to be evaluated.

5. etc.

I believe that those difficulties can be resolved and that a clear description of the state of an Algol computation will clarify the problem of compiler design.

8. Comparison with other ways of Describing Semantics

We believe that the description of programming languages by abstract syntax and state transformation functions has the following advantages.

1. Questions of notation are separated from semantic questions and postponed until the concrete syntax has to be defined.

2. Our intuitive idea of what happens when a statement is executed is described by its effect on the state.

3. This technique will lead to the most concise and understandable descriptions.

4. This notion of semantics corresponds to the notions of Tarski etc. that are current in mathematical logic. I believe that describing languages this way will lead to the possibility of proving theorems about compilers. (See the notion of correctness of a compiler presented in the ICIP paper).

It seems to me that there are two other approaches to the problem incorporated in various ways in various papers. One of these ideas is to regard the Algol data itself as strings of symbols and the state as a giant string. In my opinion this gets a long way from the intuitive ideas of Algol and enforces decisions in areas in which we want to remain uncommitted if only because of differences among machines.

A second approach is to define ALGOL by a compiler, either into a machine language,an intuitive subset of ALGOL or into an abstract system such as λ - calculus. These definitions have a certain practical value in resolving ambiguities, but as they do not correspond to our intuitive ideas, they will make mathematical results difficult to obtain and leave us with the problem of semantics of the target language.

It has been argued that since the formalism used above is a language, all semantic descriptions are circular; one might as well explain Algol by examples, etc. rather than ignore the difficulties.

In one sense this objection is unanswerable. Nothing can be explained to a stone; the reader must understand something before hand. The same objection was raised against Tarski's efforts to describe the semantics of mathematical logic which have proved very successful and fruitful.

These are two answers; a practical answer and a fundamental answer:

1. The formalism I used is simpler than Algol and will lead to understanding. The same advantage can be claimed for translations into simple languages.

2. The fundamental answer is this. The purpose of semantics is to describe the relation between the form of an expression and what it stands for. From such relations follow other properties of the example; we can define equivalence of Microalgol programs in terms of how states are transformed and show that certain changes in a program preserve equivalence.

REFERENCES

1 - Tarski, A - Logic, Semantics, and Meta-Mathematics, Oxford. -

2 - Robinson, A - Model Theory and the Meta-Mathematics of Algebra - North Holland. -

3 - McCarthy J. - Recursive Functions of Symbolic Expressions. Comm. ACM, April 1960

4 - Towards a Mathematical Science of Computation, ICIP, 1962

5 - LISP 1.5 Programmer's Manual, MIT Press, 1964